

ASSIGNMENT #5 SOLUTION

1. Program

```
; romanToDecimal takes a list of Roman numerals and returns the decimal
; equivalent if the Roman number is well formed and error otherwise.

(defun romanToDecimal (RomanNumerals)
  (accumulate RomanNumerals 0)
)

; accumulate traverses the list of Roman numerals, accumulating the decimal
; equivalent using its second argument.

(defun accumulate (RomanNumerals DecimalValue)
  (cond
    ((null RomanNumerals) DecimalValue)
    (t
     (prog (nextRomanNumeral nextNextRomanNumeral)
       (setq nextRomanNumeral (map (car RomanNumerals)))
       (cond
         ((null (cdr RomanNumerals)) (setq nextNextRomanNumeral 0))
         (t (setq nextNextRomanNumeral (map (cadr RomanNumerals))))
       )
       (cond
         ((< nextRomanNumeral nextNextRomanNumeral)
          (return
           (accumulate (cddr RomanNumerals)
            (+ DecimalValue
              (computePair nextRomanNumeral nextNextRomanNumeral))))))
         (t
          (return
           (accumulate (cdr RomanNumerals) (+ DecimalValue nextRomanNumeral))))
       )
     )
  )
)

; map converts a single Roman numeral into its decimal equivalent.

(defun map (RomanNumeral)
  (cond
    ((eq RomanNumeral 'I) 1)
    ((eq RomanNumeral 'V) 5)
    ((eq RomanNumeral 'X) 10)
    ((eq RomanNumeral 'L) 50)
    ((eq RomanNumeral 'C) 100)
    ((eq RomanNumeral 'D) 500)
    ((eq RomanNumeral 'M) 1000)
    (t 'error)
  )
)
```

```

; computePair takes 2 decimal equivalents of Roman numerals which have been
; determined to require subtraction, checks the legality of the pair, and
; performs the subtraction. Subtractions are used instead of writing four
; consecutive occurrences of the same symbol (i.e., IV should be used instead
; of IIII, MCM instead of MDCCCC, etc.). Only symbols which are powers of 10
; may be used in subtractions (i.e., I, X, C, and M). Furthermore, subtraction
; rules require that a symbol representing 10x may not precede any symbol
; larger than 10(x+1) (e.g., IC is not permitted to represent 99 but XC is 90).

```

```

(defun computePair (N1 N2)
  (cond
    ((and (power10 N1) (or (= (* N1 5) N2) (= (* N1 10) N2))) (- N2 N1))
    (t 'error)
  )
)

```

```

; power10 determines whether the argument is a power of 10 or not.

```

```

(defun power10 (N)
  (cond
    ((< N 1) nil)
    ((= N 1) 't)
    (t (power10 (/ N 10)))
  )
)

```

Output

```

>(romanToDecimal '(C D V))
405

```

```

>(romanToDecimal '(M M V I I))
2007

```

```

>(romanToDecimal '(M M M C M X C I X))
3999

```

```

>(romanToDecimal '(M I M))
Error in + [or a callee]: ERROR is not of type NUMBER.

```

2. Program

```

; matchDNA takes two strands as lists of nucleotides, and returns: 1) the
; beginning part of the second strand which does not match, 2) the part
; of the second strand which the first string matches, and 3) the remaining
; part of the second strand which does not match. If there is no match, the
; second and third components will be null. If there are multiple matching
; strings, only the first match is returned.

```

```

(defun matchDNA (strand1 strand2)
  (matchDNAstrands strand1 strand2 nil nil nil)
)

```

; matchDNAstrands is the same as matchDNA except that the 3 values to be
; returned are carried as parameters.

```
(defun matchDNAstrands (strand1 strand2 first second third)
  (cond
    ((null strand2) (list first second third))
    (t
     (prog (substrand rest)
       (setq substrand (car (substrandAndRest strand1 strand2 nil)))
       (setq rest (cadr (substrandAndRest strand1 strand2 nil)))
       (cond
         ((null substrand)
          (return (matchDNAstrands strand1 (cdr strand2)
                                   (append first (list (car strand2))) second third))
         )
         (t (return (list first substrand rest)))
       )
     )
  )
)
```

; substrandAndRest verifies that its first argument matches the second
; argument. The third argument is used to carry the part of the substrand
; matched so far.

```
(defun substrandAndRest (strand1 strand2 substrand)
  (cond
    ((null strand1) (list substrand strand2))
    ((null strand2) (list nil nil))
    ((matchPolynucleotide (car strand1) (car strand2))
     (substrandAndRest (cdr strand1) (cdr strand2)
                       (append substrand (list (car strand2))))
    )
    (t (list nil nil))
  )
)
```

; matchPolynucleotide determines whether its two arguments match or not.

```
(defun matchPolynucleotide (polynucleotide1 polynucleotide2)
  (or
    (and (eq polynucleotide1 'A) (eq polynucleotide2 'T))
    (and (eq polynucleotide1 'T) (eq polynucleotide2 'A))
    (and (eq polynucleotide1 'C) (eq polynucleotide2 'G))
    (and (eq polynucleotide1 'G) (eq polynucleotide2 'C))
  )
)
```

Output

```
>(matchDNA '(A T G C) '(A T G C A T A C G A))
((A T G C A) (T A C G) (A))
```

```

>(matchDNA '(A) '(A T G C A T))
((A) (T) (G C A T))

>(matchDNA '(C G) '(A T G C))
((A T) (G C) NIL)

>(matchDNA '(G A A T T C) '(T G G A A T T C T))
((T G G A A T T C T) NIL NIL)

```

```

3. S1   input a;
   S2   input b;
   S3   k := 1;
   S4   while (b > 0) loop
   S5     b := b - 1;
   S6     k := k * a;
         end loop;
   S7   output k

```

$$\begin{aligned}
& \underline{M[S_1; S_2; S_3; S_4; S_7]} \langle 4, 3 \rangle \\
& (S[S_1; S_2; S_3; S_4; S_7] \langle \lambda V.\perp, \langle 4, 3 \rangle, nil \rangle) \downarrow 3 \\
& (S[S_2; S_3; S_4; S_7] (S[\text{input } a] \langle \lambda V.\perp, \langle 4, 3 \rangle, nil \rangle)) \downarrow 3 \\
& (S[S_2; S_3; S_4; S_7] (\text{if } \langle 4, 3 \rangle = nil \text{ then } \top \text{ else } \langle \lambda V.\perp[hd(\langle 4, 3 \rangle)/a], tl \langle 4, 3 \rangle, nil \rangle)) \downarrow 3 \\
& (S[S_2; S_3; S_4; S_7] \langle \lambda V.\perp[hd(\langle 4, 3 \rangle)/a], \langle 3 \rangle, nil \rangle) \downarrow 3 \\
& (S[S_2; S_3; S_4; S_7] \langle \lambda V.\perp[4/a], \langle 3 \rangle, nil \rangle) \downarrow 3 \\
& (S[S_3; S_4; S_7] (S[\text{input } b] \langle \lambda V.\perp[4/a], \langle 3 \rangle, nil \rangle)) \downarrow 3 \\
& (S[S_3; S_4; S_7] \\
& \quad (\text{if } \langle 3 \rangle = nil \text{ then } \top \text{ else } \langle \lambda V.\perp[4/a][hd(\langle 3 \rangle)/b], tl \langle 3 \rangle, nil \rangle)) \downarrow 3 \\
& (S[S_3; S_4; S_7] \langle \lambda V.\perp[4/a][3/b], nil, nil \rangle) \downarrow 3 \\
& (S[S_4; S_7] (S[k := 1] \langle \lambda V.\perp[4/a][3/b], nil, nil \rangle)) \downarrow 3 \\
& (S[S_4; S_7] (\langle \lambda V.\perp[4/a][3/b][E[1] (\lambda V.\perp[4/a][3/b])/k], nil, nil \rangle)) \downarrow 3 \\
& (S[S_4; S_7] \langle \lambda V.\perp[4/a][3/b][1/k], nil, nil \rangle) \downarrow 3 \\
& (S[S_7] (S[\text{while } (b > 0) \text{ loop } S_5; S_6 \text{ end loop}] \langle \lambda V.\perp[4/a][3/b][1/k], nil, nil \rangle)) \downarrow 3 \\
& (S[S_7] \\
& \quad (\text{if } C[b > 0] (\lambda V.\perp[4/a][3/b][1/k]) \text{ then} \\
& \quad \quad \underline{S[S_4] (S[S_5; S_6] \langle \lambda V.\perp[4/a][3/b][1/k], nil, nil \rangle)} \\
& \quad \text{else} \\
& \quad \quad \langle \lambda V.\perp[4/a][3/b][1/k], nil, nil \rangle)) \downarrow 3 \\
& (S[S_7] \\
& \quad (\text{if } E[b] (\lambda V.\perp[4/a][3/b][1/k]) \langle E[0] (\lambda V.\perp[4/a][3/b][1/k]) \text{ then} \\
& \quad \quad \underline{S[S_4] (S[S_5; S_6] \langle \lambda V.\perp[4/a][3/b][1/k], nil, nil \rangle)} \\
& \quad \text{else} \\
& \quad \quad \langle \lambda V.\perp[4/a][3/b][1/k], nil, nil \rangle)) \downarrow 3
\end{aligned}$$

$(S[S_8]$
 (if $4 \geq 0$ then
 $S[S_4] (S[S_5; S_6] < \lambda V. \perp[4/a][3/b][1/k], nil, nil >)$
 else
 $< \lambda V. \perp[4/a][3/b][1/k], nil, nil >)) \downarrow 3$
 $(S[S_7] (S[S_4] (S[S_5; S_6] < \lambda V. \perp[4/a][3/b][1/k], nil, nil >))) \downarrow 3$
 $(S[S_7] (S[S_4] (S[S_6] (S[b := b - 1] < \lambda V. \perp[4/a][3/b][1/k], nil, nil >)))) \downarrow 3$
 $(S[S_7] (S[S_4] (S[S_6] (< \lambda V. \perp[4/a][3/b][1/k][0/v][E[b - 1] (\lambda V. \perp[4/a][3/b][1/k])/b], nil, nil >)))) \downarrow 3$

 $(S[S_7] (S[S_4] (S[k := k * a] < \lambda V. \perp[4/a][3/b][1/k][2/b], nil, nil >))) \downarrow 3$
 $(S[S_7] (S[S_4]$
 $(< \lambda V. \perp[4/a][3/b][1/k][2/b][E[k * a] (\lambda V. \perp[4/a][3/b][1/k][2/b])/k], nil, nil >))) \downarrow 3$
 $(S[S_7] (S[S_4] (\lambda V. \perp[4/a][3/b][1/k][2/b][4/k], nil, nil >))) \downarrow 3$
 $(S[S_7]$
 (if $C[b > 0] (\lambda V. \perp[4/a][3/b][1/k][2/b][4/k])$ then
 $S[S_4] (S[S_5; S_6] < \lambda V. \perp[4/a][3/b][1/k][2/b][4/k], nil, nil >)$
 else
 $< \lambda V. \perp[4/a][3/b][1/k][2/b][4/k], nil, nil >)) \downarrow 3$
 $(S[S_7] (S[S_4] (S[S_5; S_6] < \lambda V. \perp[4/a][3/b][1/k][2/b][4/k], nil, nil >))) \downarrow 3$
 $(S[S_7] (S[S_4] (\lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a], nil, nil >))) \downarrow 3$
 $(S[S_7]$
 (if $C[b > 0] (\lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a])$ then
 $S[S_4] (S[S_5; S_6] < \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a], nil, nil >)$
 else
 $< \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a], nil, nil >)) \downarrow 3$
 $(S[S_7] (S[S_4] (S[S_5; S_6] < \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a], nil, nil >))) \downarrow 3$
 $(S[S_7] (S[S_4] (\lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a], nil, nil >))) \downarrow 3$
 $(S[S_7]$
 (if $C[b > 0] (\lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a])$ then
 $S[S_4] (S[S_5; S_6] < \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a], nil, nil >)$
 else
 $< \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a], nil, nil >)) \downarrow 3$
 $(S[output\ k] < \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a], nil, nil >) \downarrow 3$
 (if $(\lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a])[k] = \perp$ then
 \top
 else
 $< \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a], nil,$
 $append\ nil\ (list\ (\lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a][k])) >) \downarrow 3$
 (if $64 = \perp$ then \top else $< \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a], nil, append\ nil\ (list\ 64) >)$
 $\downarrow 3$
 $(< \lambda V. \perp[4/a][3/b][1/k][2/b][4/k][1/b][16/a][0/b][64/a], nil, <64>>) \downarrow 3$
 $<64>$